2 PROGRAM TESTING

2.1 Introduction

A CREDIT program may be input to the PTS system via cards, cassette, flexible disk or console typewriter. After input the source module is held on disk. All the processors and utilities described in Part 2 of this manual read input from disk and write output to disk.

The diagram on page 2.1.2 illustrates the sequence of processes needed to develop and run an executable program from CREDIT source modules.

Each source module is processed separately by the CREDIT Translator. The Translator produces intermediate object code modules. The instructions in these modules use a byte oriented addressing system. Each module may contain references to:

- Labels in the same module
- Literals, formats, key tables and pictures in the same module and/or in the same segment.
- Labels in other CREDIT modules, in the same segment and/or in other segments.
- Assembler application modules.
- Assembler system routines.

The first three types are satisfied by the CREDIT linker.

The remaining types of reference are satisfied by the Linkage Editor. This processor builds an application load module from the following object modules:

- Object modules from the CREDIT linker.
- Assembler application modules (if referenced).
- Assembler system routines.
- CREDIT interpreter.
- CREDIT debugging program (if requested).

CREDIT code cannot be executed directly. Each instruction must be interpreted by the CREDIT Interpreter. The routines within the Interpreter actually perform the functions specified in the CREDIT code. For this reason the Interpreter is built into the Load module by the Linkage Editor.

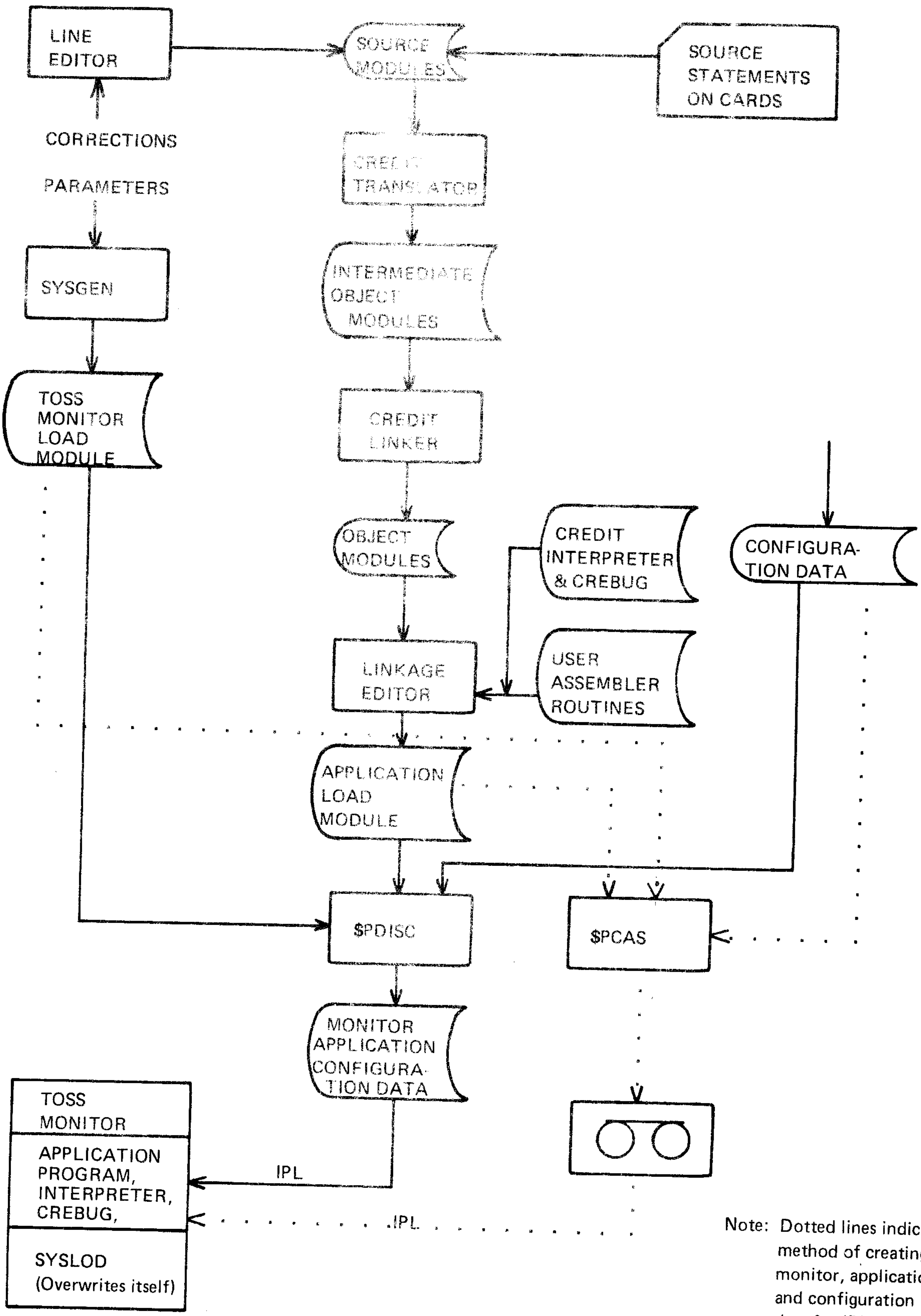
The CREDIT Debugging Program, if required, must also be built into the load module. The Debugging Program is an interactive diagnostic task which, if present, is executed in parallel with the CREDIT program being tested. Via the Debugging Program the programmer can monitor and control the execution of his program.

When the load module is loaded into memory for execution the work blocks, stacks, data set buffers and task control areas required for a particular system, are set up. This is done by the System Loading Program (SYSLOD).

The CREDIT translator, CREDIT linker and linkage editor are all run under the DOS 6800 monitor. The load module produced by the linkage editor, however, must be run under the [illegible]

[illegible] application load module, control is handed to the System Loading Program (SYSLOD). When configuration is complete, control is handed to the [illegible]. Program testing may then be carried out using one or more work positions. If the CREDIT debugging program is being used, execution can be controlled from the console typewriter.

Development of a CREDIT Application Program



Note: Dotted lines indicate method of creating monitor, application and configuration data for IPL from cassette.

If any application program errors are detected during testing, one or more source modules will have to be corrected. This may be done via the Line Editor – an interactive text editor. Each corrected source module must then be re-processed by the CREDIT Translator. The whole program must then be processed by the CREDIT Linker and Linkage Editor prior to re-testing.

TOSS system software comprises the following components:

– Monitor
– System Loading Program
– CREDIT interpreter
– [CREDIT debugging program]
– [Assembler debugging program]

These software components are not described in a separate manual. Information concerning TOSS System Software which is needed by CREDIT programmers is contained in this manual.

The following software components, though part of DOS6800 System Software, are discussed in this manual:

- CREDIT Translator
- CREDIT Linker

They are discussed in this manual because they are used by CREDIT programmers only. The remaining DOS 6800 System Software components used by CREDIT programmers, notably the Linkage Editor and Line Editor, are described in the DOS6800 System Software PRM (M11).

2.2 CREDIT Translator

2.2.1 *Introduction*

The CREDIT Translator is a processor which converts CREDIT source statements into intermediate object code. Source modules are translated separately, resulting in the production of individual object modules. References between object modules and references to external routines etc., are not resolved by the Translator.

Readers of Section 2.2 should be familiar with the following DOS6800 System Software concepts:

- Control Command
- Processor
- EOF mark
- Source input device
- Temporary source file
- Temporary object file

These concepts are explained in the DOS6800 System Software PRM (M11).

2.2.2 *Running the Translator*

Source modules must be read into the System by issuing the control command RDS (read source). RDS will read the source module from the input device (card reader, cassette or console keyboard) and will create a temporary source file. The module must be terminated by an :EOF mark. If the module has been read into the System previously and kept (control command KPF), a RDS command will not be necessary.

It is strongly recommended that all temporary object files are scratched (and kept if necessary) before the Translator is executed. This will ensure that the output object modules will not be corrupted by existing files.

The translator is called into execution by the following control command:

$$\text{TRA}\sqcup\left\{\begin{array}{l}\text{/S}\\ \text{name}\end{array}\right\}\text{[,NL]}$$

where:
- /S indicates that the input source module is in the temporary source file.
- name is the name of a source file in the library of the current user identifier. It indicates that the input source module will be found in that file.
- NL indicates that no listing of the module is required. Error messages are always printed.

The intermediate object module created by the Translator is written into the temporary object file. If this file already contains object modules, the following action is taken. If it has not been closed by an EOF mark, the intermediate object module is written *after* the information already held in the file. If it has been closed by an EOF mark, a new temporary object file is created and the old one is deleted.

2.2.3 *Translator Listing*

2.2.3.1 *General*

During translation the Translator generates a listing in three parts. Part one contains the CREDIT source statements, intermediate object code and error messages. Parts two and three contain the data item name table and the procedure label table. The following sections describe these parts.

The listing can be suppressed if the NL option is specified on the TRA control command. In this case, only the error messages will be printed.

2.2.3.2 *CREDIT code and Error Messages*

The format of this part is shown in the following example. The example is taken from the procedure division. The data division listing is slightly different. The differences are noted in the explanation which follows.

At the left of the listing under the heading LOC is the location counter. This is a four digit hexadecimal counter which is stepped by one each time a byte of intermediate object code is generated. In the data division a two digit hexadecimal counter called IX (for index) is used.

The next eight items, under the headings OC (operation code) and OPERANDS, comprise the generated interpretive instructions. Each item is a two digit hexadecimal code. The significance of these codes is described for each instruction in the Instruction Reference Section (1.4.8). Object code is not listed in the data division.

The item under the heading LINE is a four digit decimal line counter.

The remaining items are self-explanatory. They comprise CREDIT source statements.

Errors in the source module are reported by the Translator. One of the following messages is printed immediately after the line containing the error:

01	Memory overflow (job aborted)
02	Sequence error
03	Directive missing
04	Syntax error
05	Length truncation (no error accumulation)
06	Multidefined
07	Undefined
08	Unexpected value
09	Undefined type
0A	Unexpected type
0B	Illegal constant
0C	Lit pool overflow
0D	Label missing
0E	Illegal value def
0F	Illegal const length
10	Illegal const type
11	Too many blocks
12	Too many data items.
13	Block size overflow
14	Too many datasets
15	Too many parameters
16	Too many start stmts
17	Illegal dimension.
18	Too many values.
19	Out of range.
1A	Unspecified parameter.
1B	Parameterlist overflow.

In addition to the error message an asterisk is printed to show the position at which the error occurred.

An error count is maintained by the Translator and is printed after the END directive.

If a fatal error occurs (I/O error, table overflow, etc), the source input is read until an EOF mark is encountered and the following message is printed:

FATAL ERROR HAS OCCURRED. NO OBJECT CODE PRODUCED.

The object file is then deleted.

2.2.3.3 *Data Item Name Table*

The data item name table is listed immediately after the CREDIT code and error messages. For each data item declared in a work block it contains the following:

NAME — This is the data item identifier.

REF — This is the index number assigned by the Translator. Index numbers are printed to the left of the data item declarations, under the heading 'IX'.

TYPE — This is the data item type specified in the data item declaration. The following mnemonics are printed under the TYPE heading:
BCD (decimal), BIN (binary), BOL (boolean) and STR (string).
A letter U following one of these mnemonics indicates that the data item is not referenced in this module.

2.2.3.4 *Procedure Label Table*

The procedure label table is printed immediately after the data item name table. For each identifier appearing in the procedure division it contains the following:

NAME — This is an identifier specified by the programmer in the procedure division, or it is the name of a System routine referred to by the generated object code.

REF — This may be an index to a format list, a key table or an external table. It may be a value specified in an equate directive. It may be the contents of the location counter (if the identifier belongs to an instruction or PROC directive).

TYPE — This indicates the type of identifier. The following mnemonics are used:

ADR	Address of an instruction
EQU	Equate directive
EXT	External label
FOR	Format list
KEY	Key table
PRO	PROC directive
FLB	Format label (address of a format item)
FTB	Format table

A letter U following one of these mnemonics indicates that the identifier is not referred to in this module.

```
CREDIT TRANSLATOR   REL 3.1  780520 *   PROCEDURE DIVISION              IDENT  OPCLOS * 060477 *  PAGE 0006

LOC   OC  OPERANDS      LINE LABEL      OPCOD OPERANDS                  COMMENT                 C       1D
                          0336             EJECT
                          0337 *
                          0338 *
                          0339 *           ENTRY TERMINAL CLOSE MODULE
                          0340 *
                          0341 *
                          0342   OPCL05
0000   30 XX 85 FF        0343             EDWRT DSTPJT,FRM003             PRINT ASTERIX
0004   30 XX 85 FF        0344             EDWRT DSTPJT,FRM014             PRINT CLOSE
                          0345
                          0346   OPC010
0008  XX KK LL LL         0347             PERF    READN,KTAB3,=W'4',=W'4'  READ OPERID
000C  5F  06              0348             SB      OPC010
                          0349
000E   20  60 42 05       0350             CB      EQ,CASHID,INPUT,OPC100  CORRECT IDENTITY
0012  XX  12 LL           0351             PERF    ERROR,TLAMP3,KTAB4
0015   5F  0F             0352             SB      OPCO10
```

LOCATION COUNTER | LINE COUNTER | SOURCE STATEMENT LINE IMAGE | COMMENTS | CONTINUATION | IDENTIFICATION

HEXADECIMAL REPRESENTATION OF THE INSTRUCTION

RR = Reference in this module
LL = Literal constant
XX = External reference or operation code
FF = Reference to a format list
KK = Reference to a key table

CREDIT TRANSLATOR LISTING EXAMPLE

2.3 CREDIT Memory Management Linker

2.3.1 *Introduction*

The CREDIT memory management linker is a three pass processor which converts intermediate object code, produced by the CREDIT translator into object code which can be processed by the Linkage Editor. The CREDIT linker is capable of linking both unsegmented and segmented programs.

Intermediate object modules may contain references to:

- Labels in the same module.
- Literal constants, formats, key tables, pictures in the same module.
- Labels in other CREDIT modules in the same segment.
- Labels in other CREDIT modules in other segments.
- Assembler application modules.
- Assembler System routines.

The first three types of reference, when present in the same segment, are satisfied by the CREDIT linker.
The remaining types of reference must be satisfied by the Linkage Editor.
To build up the segments, different possibilities exist which are the same when using extended main memory, secondary memory or a combination of both.

Readers should be familiar with the following DOS6800 System Software concepts:

- Control command
- User library
- User identifier
- Temporary object file

These concepts are explained in the DOS6800 System Software PRM (M11).

2.3.2 *Building up segments*

After translation of the different CREDIT modules a number of intermediate object modules have been created. All these modules together building up a CREDIT application, are input to the CREDIT linker (TLK).
The CREDIT linker has to know which modules should be contained in the segments. This is specified by the user by means of the ordering of the modules as input to the linker. In the TLK command is a parameter (n or mK) defining the maximum segment size to be used.
However, the NOD command (node) can be used to force an immediate end of a segment and also to define the segment as main memory resident (NOD ␣ R) or belonging to the common area, segment 00 (NOD ␣ C). The common area, segment 00, is always present in main memory and will contain the data division, the interpreter, assembler subroutines and / or user routines. The size of the common area is variable and not dependent on the size parameter in the TLK command. Segment 00 is automatically created.
When the NOD command is used without specifying R or C as parameter, the segment will be disk resident. When the NOD command is not used, the segments will be disk resident.

The output from the CREDIT linker is placed in temporary object file (/0) and divided into different modules as: data division, common part and segments. The user can extend the common part, segment 00, with the NOD ␣ C command. The modules are grouped into segments in the order in which they are included with the INC command. The segments are numbered from 1 upward.

Some examples showing the use of NOD and TLK for different systems.

a. System with 64K Byte main memory.

```
INC   MOD1
INC   MOD2
INC   MOD3
TLK   U,M,X
```

b. System with 64K Byte main memory and use of secondary memory. (Disk, flexible disk).

```
                                    Size
INC   MOD1                          (3.5Kbytes)
INC   MOD2,   USER1                 (0.7Kbytes)
INC   MOD3,   USER2                 (0.5Kbytes)
INC   MOD4                          (2Kbytes)
INC   MOD5                          (2Kbytes)
INC   MOD6,   USER3                 (1Kbyte)
INC   MOD7                          (2Kbytes)
TLK   U,M,4K
```

Four segments are created by the linker, and all are disc resident. (Segment zero always resides in main memory.)

Segment 1	Contains	MOD1	(3.5K)
Segment 2	Contains	MOD2, MOD3, MOD4	(3.2K)
Segment 3	Contains	MOD5, MOD6	(3K)
Segment 4	Contains	MOD7	(2K)

By means of altering the sequence of the INC commands, the user can optimize his program segments. In this example only 50% of sement 4 is filled.
When e.g. MOD5 must be present in segment 0, the following sequence of commands has to be specified:

```
                                    Size
NOD   C
INC   MOD5                          (2Kbytes)
NOD
INC   MOD1                          (3.5Kbytes)
INC   MOD2,   USER1                 (0.7Kbytes)
INC   MOD3,   USER2                 (0.5Kbytes)
INC   MOD4                          (2Kbytes)
INC   MOD6,   USER3                 (1Kbyte)
INC   MOD7                          (2Kbytes)
TLK   U,M,4K
```

Segment 1 contains:	MOD1
Segment 2 contains:	MOD2, MOD3, MOD4
Segment 3 contains:	MOD6, MOD7,.

MOD5 is now included in the common area, segment zero.

When also MOD1 must be main memory resident, but not in segment 0, then the following command sequence can be used:

			Size
NOD	C		(2Kbytes)
INC	MOD5		(2Kbytes
NOD	R		
INC	MOD1		(3.5Kbytes)
NOD			
INC	MOD2,	USER1	(0.7Kbytes)
INC	MOD3,	USER2	(0.5Kbytes)
INC	MOD4		(2Kbytes)
INC	MOD6,	USER3	(1Kbyte)
INC	MOD7		(2Kbytes)
TLK	U,M,4K		

Segment 1 contains:	MOD1	(3.5Kbytes)
Segment 2 contains:	MOD2,MOD3,MOD4	(3.2Kbytes)
Segment 3 contains:	MOD6, MOD7	(3Kbytes)

c. System with extended main memory, up to 256Kbytes.

			Size
INC	MOD1		(3.5Kbytes)
INC	MOD2,	USER1	(0.7Kbytes)
INC	MOD3,	USER2	(0.5Kbytes)
INC	MOD4		(2Kbytes)
INC	MOD5		(2Kbytes)
INC	MOD6,	USER3	(1Kbyte)
INC	MOD7		(2Kbytes)
TLK	U,M,4K		

Four segments are created by the linker, and are assumed to be disk resident. Because an extended main memory is used, all segments will be main memory resident. The composition of the segments is as mentioned in example b.
The system loader SYSLOD will discover the difference when a system with extended main memory, secondary memory or a combination of both is used.

d. Systems with extended main memory (up to 256 bytes) and secondary memory.

Examples b) and c) may be combined.

2.3.3 *Running linker*

The CREDIT linker reads intermediate object modules from temporary object file and from the library of the current user identifier. The syntax of the TLK command is:

TLK ␣ [N|S|U][,X] [,M] [,n|,mK]

N The system or user /ØBJCT files do not need to be scanned.

U Only the user /ØBJCT files will be scanned.
S Only the system /ØBJCT file has to be scanned.

Default value: Both /ØBJCT files will be scanned.
The user /ØBJCT file will be scanned first, then the system /ØBJCT file and then the user /ØBJCT file again.

X Indicates that a cross reference listing is required.

Default value: No cross reference will be printed.

M The listing of the map, which consists of a listing of the module names and the relative start addresses, address pools and statistics per segment.

Default value: No map will be printed.

n The required segment size in bytes.
mK The required segment size in K bytes. (m x 1024 bytes)

Default value: The program will be unsegmented.

2.3.3.1 *CREDIT modules in the system library*

When CREDIT modules have to be linked from the System library (USERID:SYSTEM), first a Generate Object Directory command (GOD) must be executed for this library before the TLK command can be executed. (In any other user than SYSTEM.)

The following listings are produced by the CREDIT memory management linker per segment:

– Segment 0 (Common part)
  - load map
  - long branch table
  - call table
  - perform table
  - literal pool
  - key table pool
  - picture pool
  - format pool

  (long branch table … format pool: can be excluded by not using 'M' in the TLK command.)

  Linker statics for this segment

– Segment n
  - loadmap
  - long branch table
  - perform table
  - literal pool
  - picture pool
  - format pool

  (long branch table … format pool: can be excluded by not using 'M' in the TLK command.)

  Linker statics for this segment

– Total
  - segment map
  - cross reference

  (can be excluded by not using 'X' in the TLK command.)

  Linker statics for the whole program.

The load map includes a list of error reports. Error reports will be listed even if the load map listing has not been requested.

2.3.3.2 *Load Map*

The load map indicates the displacement of each module within a segment. It also contains the linker (TLK) error reports. The format of the load map is shown in the following example:

```
------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * LOAD MAP  SEGMENT  02
------------------------------------------------------------


LOC       MODULE    ERROR          COMMENT

000E      MOD3                     TRA 4.1   99-99-99 F1 01111
006D      MODUL4                   TRA 4.1   99-99-99 F1 01111
0090      MODUL6                   TRA 4.1   99-99-99 F1 01111
00B9      MODUL7                   TRA 4.1   99-99-99 F1 01111
00C8      MODUL8                   TRA 4.1   99-99-99 F1 01111
```

where: LOC is the displacement of the module within the segment.
MODULE is the module name.
ERROR is the error number followed by a type, number and clear text.

Error type may be:

E — User Error
I — Internal error or input inconsistency
W — Warning, no updating of error counter.

The following error reports may be printed:

ERROR NUMBER	ERROR TYPE	Additional Information	Text (Significance)
0	I		END OF MEMORY No more work space available
1	E		SYMBOL TYPE CONFLICT LB, CALL or PERF mixed up
2	I	XXXX	ILLEGAL INPUT XXXX is a hexadecimal presentation of 1st and 3rd character in cluster. Input from translator not expected.
3	I	XXXX	LOAD ADR INCORRECT XXXX is a hexadecimal presentation of load address from the cluster.
4	W	DDDD	UNREFERENCED LITERAL DDDD is a decimal presentation of the number of unreferenced literals.

ERROR NUMBER	ERROR TYPE	Additional Information	Text (Significance)
5	I	DDDD	UNDEFINED LITERAL DDDD is a decimal presentation of the number of undefined literals.
6	E		NO START ADR Start address declaration not found in the data division.
7	E		DBL DEF MODULES Double defined modules
8	E	DDDD	UNSATISFIED EXTERNAL DDDD is a decimal presentation of the number of unsatisfied externals. The unsatisfied externals are printed on the load map.
9	I	XXXX	MODULE LENGTH ERROR XXXX is a hexadecimal presentation of the difference between requested and available workspace.
10	E	DDDD	TRANSLATION ERROR DDDD is a decimal presentation of the number of translating errors.
11	E	XXXX	WRONG TRANSLATOR RELEASE XXXX is a hexadecimal representation of the lowest acceptable level of the CREDIT translator, in the form RRLL RR = Release number LL = Level number
12	E	C	ADDRESS TABLE OVERFLOW C is a character representing the address type L (Long Branch), C (Call) or P (Perform).
13	E	C	LITERAL DISPLACEMENT OVERFLOW C is a character representing the literal type L (Literal), K (key table), P (picture), or F (format).
14	E	C	TOO MANY LITERALS C is a character representing the literal type L (literal), K (key table), P (picture), or F (format).
15	E		FORMAT LENGTH ERROR
16	E		MULTI DEF ENTRY Entry name defined in more than one module.
17	E	C	NOD TYPE ERROR C is a character representing the NOD type. NOD type not C, D or R.

ERROR NUMBER	ERROR TYPE	Additional Information	Text (Significance)
18	E	XXXX	MODULE LONGER THAN SEGMENT SIZE XXXX is a hexadecimal representation of the module length. Increase segment size.
19	I		IDENT MISSING
20	E		ADDRESSING MODE CONFLICT One byte or two bytes addressing mode of literals mixed up.
21	E		NOD SEQUENCE ERROR The NON record "NOD C" does not appear in the beginning of the object input.

2.3.3.3 *Call table*

This table contains all references to external routines (CALL instruction) which could not be satisfied by the TLK command. Each time a reference is encountered in the intermediate code, the linkage editor (LKE command), replaces it by an "index value" which points to the called address in the call table. During execution of the application program, the interpreter refers to the call table for actual destination addresses. The format of the call table is shown in the following example:

```
------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * CALL TABLE   SEGMENT  00
------------------------------------------------------------------

LOC   DATA      IX  SYMBOL   DEFINED

0002  ****      01  T:ASSI
0004  ****      02  T:KI
0006  ****      03  T:EDWR
0008  ****      04  T:DSC1
000A  ****      05  T:NKI
000C  ****      06  T:RREA
000E  ****      07  T:RWRI
```

where: LOC — is the displacement of each table entry within segment zero.
DATA — is the called address relative to the start of segment zero. It is generated by the [illegible] editor and is therefore not specified in the listing.
IX — is the index value [illegible] (maximum index is X'FF')
SYMBOL — is the name of the external routine.
DEFINED — is not used in this table.

2.3.3.4 *Long branch table*

In order to reduce the amount of memory required for a long branch instruction, linker (TLK) generates a table of destination addresses. Each time a long branch is encountered in the intermediate code, the linker places the destination address (i.e. segment number and the address to be branched to) in the long branch table.
The three byte destination address in the long branch instruction is replaced by a one byte "index value" which points to the destination address in the long branch table. During execution of the application program the interpreter refers to the long branch table for actual destination addresses. The format of the long branch table is shown in the following example:

```
------------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * LB TABLE  SEGMENT  01
------------------------------------------------------------------------------

LOC   DATA     IX  SYMBOL  DEFINED

0170  01 007A  01          MODUL5
0174  01 0054  02          MODUL5
0178  01 011B  03          MODUL5
017C  01 009D  04          MODUL5
0180  01 00F8  05          MODUL5
0184  01 00E3  06          MODUL5
0188  01 00C0  07          MODUL5
018C  01 010F  08          MODUL5
```

where: LOC — is the displacement of each table entry within the segment.
DATA — is the destination addres of the long branch. The first two digits specify the segment number and the next four specify the displacement within this segment. The difference between the four digit hexadecimal value, and the relevant module start address shown in the load map, gives the address of the destination within that module.

IX	is the index value used in the long branch instructions. It starts at one. (Maximum index is X'FF').
SYMBOL	is the statement identifier of the first instruction (location counter = 0) in the module containing the destination of the branch.
DEFINED	is the name of the module containing the destination.

2.3.3.5 *Perform table*

This table contains the address of each CREDIT subroutine which is called (PERF instruction) within this segment. It has the same layout as the long branch table. Each time a perform to a CREDIT subroutine is encountered, in the intermediate object code the subroutine name is replaced by an "index value" which points to the subroutine address in the perform table. The format of the perform table is shown in the following example.

```
------------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * PERFORM TABLE   SEGMENT 01
------------------------------------------------------------------------------

LOC    DATA       IX   SYMBOL   DEFINED

0192   01 0037    01   VDU1     MOD1
0196   XX XXXX    02   KB1
019A   XX XXXX    03   VDU2
019E   01 0052    04   KB2      MODUL5
01A2   XX XXXX    05   DISC
01A6   01 0030    06   STP1     MOD2
01AA   XX XXXX    07   VDU3
01AE   XX XXXX    08   VDU4
01B2   01 0046    09   STP2     MOD2
```

where:	
LOC	is the displacement of each table entry within the segment.
DATA	is the destination address of the perform. The first two digits specify the segment number and the next four specify the displacement within this segment. The difference between the four digit hexadecimal value, and the relevant module start address shown in the loadmap, gives the address of the destination within that module.

IX — is the index value. It starts at one (maximum index is X'FF').
SYMBOL — is the name of the subroutine. It only appears when the subroutine is not in the same module as the perform instruction.
DEFINED — is the name of the module which contains the subroutine.

2.3.3.6 *Literal pool*

The literal pool contains all the literals used in this segment. Each time a literal is encountered in the intermediate code it is replaced by an "index value" which points to the literal in the literal pool. The format of the literal pool is shown in the following example:

```
------------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * LITERAL POOL   SEGMENT   02
------------------------------------------------------------------------------


IX     TYPE    LOC    DATA

10     BIN     00D8   0000
11     BIN     00DA   0004
12     BIN     00DC   0006
13     BIN     00DE   0008
14     BIN     00E0   0009
15     BIN     00E2   0016
16     BIN     00E4   0033
17     BIN     00E6   0040
18     BIN     00E8   0042
19     BIN     00EA   0200
1A     BIN     00EC   0410
1B     BIN     00EE   1410
1C     STR     00F0   07
1D     STR     00F1   202B
1E     STR     00F3   2030
1F     STR     00F5   203107
20     STR     00F8   4141411E
21     STR     00FC   4141421E
22     STR     0100   4141431E
23     STR     0104   4141441E
24     STR     0108   4141451E
```

where: IX — is the index value. It starts at 10 or 4100 (maximum index is X'FF' or X'4FFF').
TYPE — indicates the value type of the literal. The following mnemonics are used:
BIN for value types X and W.
BCD for value type D.
STR for value type C.
LOC — is the displacement of each literal within the segment.
DATA — is the hexadecimal representation of the literal.

2.3.3.7 *Picture pool*

The picture pool contains all picture strings used in this segment. Each time a reference to a picture string is encountered in the intermediate code, it is replaced by an "index value" which points to the picture string in the pool. The format of the picture pool is shown in the following example:

```
----------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * PICTURE POOL   SEGMENT  01
----------------------------------------------------------------------

IX     TYPE     LOC    DATA

10     PIC      01D1   393939
11     PIC      01D4   5A5A5A5A5A5A392C3939
12     PIC      01DF   3939452D3939452D39393939
13     PIC      01EB   5A5A565A5A5A565A5A392C39392B
```

where: IX — is the index value. It starts at 10 or 5100 (maximum index is X'FF' or X'5FFF').
TYPE — indicates the entry is a picture string (PIC).
LOC — is the displacement of each picture string within the segment.
DATA — is the hexadecimal representation of the picture string.

2.3.3.8 *Keytable pool*

The keytable pool contains all keytables used in the application program and is located in segment zero. Each time a reference to a keytable is encountered in the intermediate code, it is replaced by an "index value" which points to the keytable in the pool. The format of the keytable pool is shown in the following example:

```
----------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * KEYTABLE POOL   SEGMENT  00
----------------------------------------------------------------------

IX     TYPE     LOC    DATA

10     KEY      0010   031E1D19
```

where:		
	IX	is the index value. It starts at 10 or 6100 (maximum index is X'FF' or X'7FFF').
	TYPE	indicates the entry in a keytable. (KEY)
	LOC	is the displacement of the keytable within segment zero.
	DATA	is the hexadecimal representation of the keytable. First character in the keytable is the length indicator.

2.3.3.9 *Format pool*

The format pool contains all format lists used in the segment.
Each time a reference to a format list is encountered in the intermediate code, it is replaced by an "index value" which points to the format list in the pool. The format of the format pool is shown in the following example:

```
----------------------------------------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * FORMAT POOL   SEGMENT  01                  * DATE 2857   * PAGE    9 *
----------------------------------------------------------------------------------------------------------

IX     TYPE     LOC    DATA

10     FMT      01F9   C11BC027
11     FMT      01FD   C11BC026
12     FMT      0201   C11BC025
13     FMT      0205   C11B1322
14     FMT      0209   C11CC30F415554484F524954592E2E2E2E2E3A9E20C30544415444453A
15     FMT      0225   C11C9420C3192A2A5452414E53414354494F4E2043414E43454C4C45442A2AE8C11CA82AE8C11CA82A
16     FMT      024E   C11CA82AE8C11CC31F454E44204F4620444159 2C534552564943452044495343 4F4E54494E554544E8C11CA82A
                       E8C11CA82A
17     FMT      0280   C11CC30F5452414E534143544 94F4E2E2E2E3A10219E20C30544415444 53A1220E8C11CC11CC30E4F5045524154
                       4F5220434F44453AC02A9E20C30D4F464649434520 4E4F3A31323 3E8C11CC3054E414D453AC0258620C3074144
                       524553533AC026E8C11CC305434954593AC0278920C30B41434 34F554E54204E4F3AC029E8C11CC307414D4F55
                       4E543A11229020C30C4E4557204241 4C414E43453A1123
```

where:		
	IX	is the index value. It starts at 10 or 7100 (maximum index is X'FF' or X'7FFF').
	TYPE	indicates that the entry is a format list (FMT) or format table FTB). The layout of FMT entries is explained below.
	LOC	is the displacement of each format list in the pool within the segment.
	DATA	is the hexadecimal representation of the format list or format table.

Each word in an FMT entry has the following layout:

<table>
<tr><td>bit 0</td><td>1 2 3 4 5 6 7</td><td>8 9 10 11 12 13 14 15</td></tr>
<tr><td rowspan="2">T1</td><td>P</td><td rowspan="2">A</td></tr>
<tr><td>T2 0</td></tr>
</table>

Depending on the T1 and T2 bit, fields P and A or 0 and A have the following meaning:

T1 = 0; P field contains an index to a picture string (FMEL).
A field refers to decimal-data-item

T1 = 1; O field contains a six-bit value, indicating how many times the character in the A-field has to be copied. (FILLR).

T1 = 1, T2 = 1;

Contents 0-field	Significance
00/01	A-field contains a reference to a string-data-item or literal (FCOPY).
03	A-field and following bytes contain ISO-7 characters (FTEXT).
04	A-field contains a tabulation value. (FTAB).
08	Character X'1F' edited into the buffer. A-field not used (FHIGH).
09	Character X'1E' edited into the buffer. A-field not used (FLOW).
0A	Character X'12' edited into the buffer. A-field not used (FUL).
0B	Character X'13' edited into the buffer. A-field not used (FNUL).
10	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement. (FBZ).
11	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement. (FBP).
12	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement (FBN).
14	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement (FBNZ).
15	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement (FBNP).
16	A-field contains a reference to a binary or decimal-data-item. The byte following the A-field contains a displacement (FBNN)
18	A-field contains a displacement (FB).
1A	A-field contains a reference to a boolean-data-item. The byte following the A-field contains a displacement (FBF).
1B	A-field contains a reference to a boolean-data-item. The byte following the A-field contains a displacement (FBT).
1C	A-field contains a reference to a binary-data-item. (FCW).
1D	A-field contains a reference to a literal (FCW).
1F	A-field contains a reference to a subformat list (FLINK).
20	A-field not used (FSL).
21	A-field not used (FNL).
28	A-field not used (FEOR).
29	A-field not used (FEXIT).

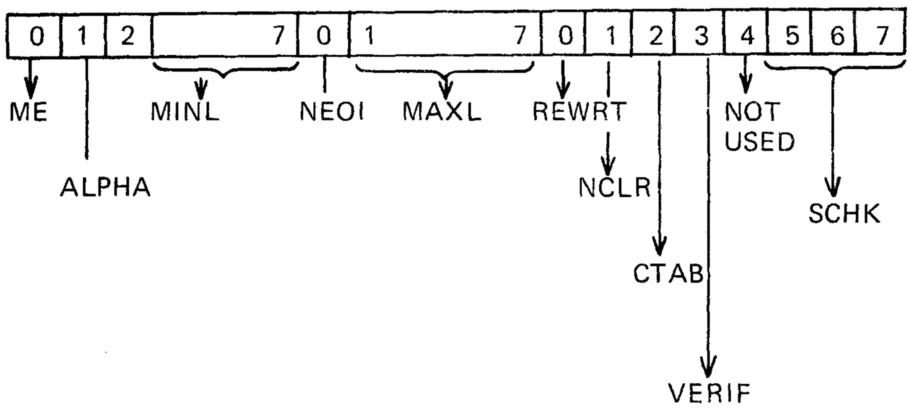
2C A-field contains the tabulation position (FINP).

2E A-field contains the tabulation position, the following byte contains the right halfword of the application (APPL) control field (FINP).

2F A-field contains the tabulation position, the following 2 bytes contain in sequence:
a) left halfword of the application (APPL) control field
b) right halfword of the application (APPL) control field (FINP).

30 A-field contains the tabulation position. The three following bytes constitute the standard input control field (FKI).

Layout standard control field (FKI).



32 A-field contains the tabulation position. The following byte contains the right halfword of the application (APPL) control field.
The next three bytes are the standard control field bytes (see also 30) (FKI).

33 A-field contains the tabulation position. The following bytes contain in sequence:
a) left halfword of the application (APPL) control field
b) right halfword of the application (APPL) control field
c), d) and e) are the standard control field bytes (see also 30) (FKI).

38 A-field contains the tabulation position. The following bytes contain in sequence :
a) duplication data-item (DUPL) reference.
b), c) and d) are the standard control field bytes (see also 30) (FKI).

3A A-field contains the tabulation position. The following bytes contain in sequence:
a) right halfword of the application (APPL) control field
b) duplication data-item (DUPL) reference
c), d) and e) are the standard control field bytes (see also 30) (FKI).

3B — A-field contains the tabulation position the following bytes contain in sequence:

a) left halfword of the application (APPL) control field
b) right halfword of the application (APPL) control field
c) duplication data-item (DUPL) reference
d), e) and f) are the standard control field bytes (see also 30) (FKI).

2.3.3.10 *Linker statistics per segment*

The format of the linker statistics listing per segment, is shown in the following example. The contents of the listing are self-explanatory.

```
-------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * LINKER STATISTICS   SEGMENT 00
-------------------------------------------------------------------------

 ALL VALUES DECIMAL

                     LB TABLE:      0 BYTES.     0 ENTRIES
                   CALL TABLE:     14 BYTES.     7 ENTRIES
                PERFORM TABLE:      0 BYTES.     0 ENTRIES

 LITERAL DESCRIPTOR TABLE:      0 BYTES.     0 ENTRIES
 PICTURE DESCRIPTOR TABLE:      0 BYTES.     0 ENTRIES
KEYTABLE DESCRIPTOR TABLE:      4 BYTES.     1 ENTRIES
  FORMAT DESCRIPTOR TABLE:      0 BYTES.     0 ENTRIES

        LITERAL POOL SIZE:      0 BYTES
        PICTURE POOL SIZE:      0 BYTES
       KEYTABLE POOL SIZE:      4 BYTES
         FORMAT POOL SIZE:      0 BYTES

  INTERPRETABLE CODE SIZE:      0 BYTES
           PROGRAM LENGTH:     40 BYTES

         NUMBER OF ERRORS       0
```

2.3.3.11 *Segment map*

This map gives a listing of the number of segments, the number of modules contained in a segment and the number of bytes per segment. The format of the segment map is shown in the following example:

```
---------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * SEGMENT MAP
---------------------------------------------------------

   S E G M E N T                          N U M B E R   O F
NUMBER   TYPE  LENGTH     USAGE          MODULES     ERRORS

00        C        40                          0          0
01        D       914      91 %                4          0
```

where: NUMBER is the segment number.
TYPE indicates:
C = common part (segment zero)
R = main memory resident
D = disk resident

LENGTH	number of bytes contained in this segment (program length).
USAGE	a filling percentage of the segment, related to the size option in the TLK command.
MODULES	number of modules contained in the segment.
ERRORS	number of errors per segment.

2.3.3.12 *Address cross reference listing*

This listing provides a cross reference between statement/subroutine identifiers in the procedure division and the modules/segments in which they are referenced. The format of the address cross reference listing is shown in the following example:

```
-----------------------------------------------------------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * CROSS REFERENCE LISTING                                * DATE 2857   * PAGE
-----------------------------------------------------------------------------------------------------------------

SYMBOL  TYPE  VALUE    SEG-DEFINED    REFERENCES

DISC    P     02 0090  02-MODUL6      01-MAIN    (1)
GO      B S   01 000E  01-MAIN        02-MODUL6 (1)    02-MODUL7 (1)    02-MODUL8 (1)
GTP1    P     01 0030  01-MOD2        01-MAIN    (1)
GTP2    P     01 0046  01-MOD2        01-MAIN    (1)
GTP3          01 004C  01-MOD2
KB1     P     02 000E  02-MOD3        01-MAIN    (1)
KB2     P     01 0052  01-MODUL5      01-MAIN    (1)
T:ASSI  C                             01-MAIN    (1)
T:DSC1  C                             01-MODUL5 (8)    02-MOD3    (4)    02-MODUL4 (3)    02-MODUL6 (1)
                                      02-MODUL7 (1)
T:EDWR  C                             01-MOD1    (1)    01-MOD2    (3)    01-MODUL5 (4)    02-MOD3    (1)
                                      02-MODUL4 (1)    02-MODUL6 (1)    02-MODUL7 (1)    02-MODUL8 (1)
T:KI    C                             01-MAIN    (1)    01-MODUL5 (3)    02-MOD3    (2)
T:NKI   C                             01-MODUL5 (2)
T:RREA  C                             01-MODUL5 (1)
T:RWRI  C                             02-MODUL6 (1)
VDU1    P     01 0037  01-MOD1        01-MAIN    (1)
VDU2    P     02 0060  02-MODUL4      01-MAIN    (1)
VDU3    P     02 00B9  02-MODUL7      01-MAIN    (1)
VDU4    P     02 00C8  02-MODUL8      01-MAIN    (1)
```

where:

SYMBOL	is the statement/subroutine identifier in the procedure division.
TYPE	indicates type of instruction in which "symbol" is used. C = CALL P = Perform B = Branch S = Start point.
VALUE	displacement of "symbol" in the referenced segment. The first two digits specify the segment number and the next four specify the displacement within this segment.

SEG-DEFINED	segment number and module name which contains "symbol".
REFERENCES	are the segment numbers and module names, containing references to "symbol". The number of references in each module appears in brackets after the module name.

2.3.3.13 *Linker statistics total*

The format of the linker statistics total is shown in the following example. The contents of the listing are self-explanatory.

```
---------------------------------------------------------------
* CREDIT CODE LINKER PRR 4.1 790410 * LINKER STATISTICS TOTAL
---------------------------------------------------------------


  ALL VALUES DECIMAL

   INTERPRETABLE CODE SIZE:   547 BYTES
           PROGRAM LENGTH:  1568 BYTES

       AVAILABLE WORKSPACE: 23874 BYTES
            USED WORKSPACE:  2934 BYTES,     12 %
          UNUSED WORKSPACE: 20940 BYTES
  MAX WORKSPACE PER MODULE:   370 BYTES

          NUMBER OF ERRORS      0

PROG ELAPSED TIME:  00H-02M-01S-520MS-
```